



*Project documentation  
And  
Submission phase*

## PHASE 5: DOCUMENTATION

Data set link :

<https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset>

Description of data set:

*The "Microsoft Lifetime Stocks Dataset" available on Kaggle contains historical data related to Microsoft Corporation's stock prices. This dataset provides a comprehensive record of Microsoft's stock performance over a significant time period. Below is a description of the dataset based on common characteristics typically found in financial time series data:*

Columns/Attributes:

**Date:** *This column represents the date on which the stock prices were recorded. It serves as the primary time index for the data.*

**Open:** *The opening price of Microsoft stock on the recorded date.*

**High:** *The highest price of Microsoft stock during the recorded date.*

**Low:** *The lowest price of Microsoft stock during the recorded date.*

**Close:** *The closing price of Microsoft stock on the recorded date. This is often the most important column for stock price prediction.*



**Volume:** *The trading volume, indicating the number of Microsoft shares traded on the recorded date.*

**Adjusted Close:** *This may represent the adjusted closing price, which accounts for corporate actions like dividends and stock splits.*

### Data Granularity:

*The dataset records daily stock price data, which means that each row corresponds to a single day of trading.*

### Date Range:

*The dataset encompasses a "lifetime" of historical stock data for Microsoft. The specific date range covered by the dataset will depend on the source but could span several years or decades.*

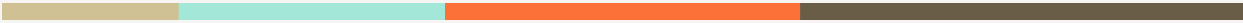
### Data Consistency:

*It is essential to ensure that the data is consistent, free of duplicates, and follows a continuous chronological order. Inconsistent data can lead to issues during analysis and modeling.*

### Missing Values:

*Data may contain missing values, particularly in the case of financial data. Handling missing data is important during data preprocessing.*

### Dataset Purpose:



*The dataset is designed for time series analysis and stock price prediction. It can be used to build predictive models to forecast future stock prices or perform various financial analyses.*

### **Usage and Applications:**

*Traders, investors, financial analysts, and data scientists can use this type of dataset for various purposes, such as trend analysis, volatility assessment, and predictive modeling for trading strategies.*

### **Outline of the problem statement:**

*The problem is to develop a predictive model for Microsoft stock prices. Given historical data, the goal is to create a model that can accurately forecast the future closing prices of Microsoft's stock. Accurate stock price predictions are valuable for investors and traders to make informed decisions.*

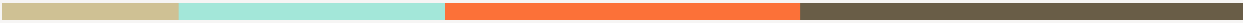
### **Design Thinking Process:**

*The design thinking process involves understanding the problem, ideating solutions, and iteratively developing a solution while keeping the end-users' needs in mind. Here's how the design thinking process can be applied to this problem:*

#### **➤ Empathize:**

- *Understand the needs and goals of the end-users (e.g., investors, traders).*

- *Identify pain points and challenges they face when making stock-related decisions.*
- **Define:**
  - *Clearly define the problem statement, specifying that the goal is to predict Microsoft stock prices.*
  - *Identify success criteria, such as the accuracy of predictions or the choice of evaluation metrics (e.g., RMSE, MAE).*
- **Ideate:**
  - *Brainstorm potential approaches and features for stock price prediction.*
  - *Consider different machine learning algorithms, data sources, and technical indicators.*
  - *Explore how the prediction model could be used in real-world scenarios.*
- **Prototype:**
  - *Implement a prototype or proof-of-concept using the code provided.*
  - *Experiment with various features and hyperparameters to improve model performance.*
- **Test:**

- 
- *Evaluate the prototype's performance using relevant evaluation metrics.*
  - *Gather feedback from potential users or stakeholders to understand their preferences and requirements.*
  - **Iterate:**
    - *Based on feedback and test results, make iterative improvements to the model and code.*
    - *Adjust the feature engineering and model selection as necessary.*
  - **Implement:**
    - *Develop a robust and scalable version of the solution.*
    - *Consider deployment options, such as creating a web application or integrating the model into a trading platform.*
  - **Deliver:**
    - *Provide the solution to end-users or stakeholders.*
    - *Offer training and support as needed.*
    - *Monitor the model's performance in real-time and make updates as necessary.*

## Phases of Development:

*The development process for a stock price prediction model can be broken down into several phases:*

### ➤ Data Collection:

- *Gather historical stock price data for Microsoft from reliable sources (e.g., financial APIs, data providers).*
- *Ensure the data is clean and contains relevant information.*

### ➤ Data Preprocessing:

- *Handle missing values, outliers, and anomalies.*
- *Convert and format data, such as date columns, as needed.*
- *Normalize or scale features.*

### ➤ Feature Engineering:

- *Create features that capture relevant information for stock price prediction, such as moving averages, technical indicators, and lagged features.*
- *Experiment with different features and analyze their importance.*

### ➤ Model Selection:

- *Choose an appropriate machine learning algorithm for regression, considering algorithms like XGBoost, LSTM, or others.*
- *Optimize hyperparameters for the selected algorithm.*
- **Model Training:**
  - *Split the data into training and testing sets.*
  - *Train the model on historical data.*
- **Model Evaluation:**
  - *Evaluate the model's performance using suitable metrics (e.g., RMSE, MAE, R<sup>2</sup>).*
  - *Fine-tune the model if needed.*
- **Deployment:**
  - *Deploy the model to a production environment if applicable.*
  - *Set up automated data collection and prediction processes.*
- **Monitoring and Maintenance:**
  - *Continuously monitor the model's performance and retrain as necessary.*
  - *Implement updates based on market dynamics or changes in user requirements.*



### ➤ User Training and Support:

- *Provide training to users if the model is used by individuals or teams.*
- *Offer support for any issues or questions.*

### ➤ Feedback and Improvement:

- *Gather feedback from users and stakeholders.*
- *Use feedback to make iterative improvements to the model, code, and user interface.*

*By following this structured development process and incorporating design thinking principles, you can create a reliable and user-centric solution for predicting Microsoft stock prices. This approach ensures that the solution meets the needs of the target audience and can adapt to changing market conditions.*

## Description of data set used:

*The "Microsoft Lifetime Stocks Dataset" available on Kaggle contains historical data related to Microsoft Corporation's stock prices. This dataset provides a comprehensive record of Microsoft's stock performance over a significant time period. Below is a description of the dataset based on common characteristics typically found in financial time series data:*

### ➤ Columns/Attributes:

- *Date: This column represents the date on which the stock prices were recorded. It serves as the primary time index for the data.*

- *Open*: The opening price of Microsoft stock on the recorded date.
  - *High*: The highest price of Microsoft stock during the recorded date.
  - *Low*: The lowest price of Microsoft stock during the recorded date.
  - *Close*: The closing price of Microsoft stock on the recorded date. This is often the most important column for stock price prediction.
  - *Volume*: The trading volume, indicating the number of Microsoft shares traded on the recorded date.
  - *Adjusted Close*: This may represent the adjusted closing price, which accounts for corporate actions like dividends and stock splits.
- **Data Granularity:**
- *The dataset records daily stock price data, which means that each row corresponds to a single day of trading.*
- **Date Range:**
- *The dataset encompasses a "lifetime" of historical stock data for Microsoft. The specific date range covered by the dataset will depend on the source but could span several years or decades.*
- **Data Consistency:**
- *It is essential to ensure that the data is consistent, free of duplicates, and follows a continuous chronological order. Inconsistent data can lead to issues during analysis and modeling.*

➤ **Missing Values:**

- *Data may contain missing values, particularly in the case of financial data. Handling missing data is important during data preprocessing.*

➤ **Dataset Purpose:**

- *The dataset is designed for time series analysis and stock price prediction. It can be used to build predictive models to forecast future stock prices or perform various financial analyses.*

➤ **Usage and Applications:**

- *Traders, investors, financial analysts, and data scientists can use this type of dataset for various purposes, such as trend analysis, volatility assessment, and predictive modeling for trading strategies.*

## Data preprocessing steps:

*Data preprocessing is essential to clean and organize the dataset for analysis and modeling. The code includes several data preprocessing steps:*

- **Conversion of the 'Date' column:** *The 'Date' column is converted to a datetime object to facilitate time-based analysis.*
- **Sorting by date:** *The dataset is sorted in ascending order based on the 'Date' column, ensuring that the data is arranged chronologically.*

- **Handling missing values:** Rows with missing values (NaN) are removed from the dataset. Missing data can cause issues during modeling and analysis, so it's essential to handle them appropriately.

## Feature Extraction Techniques:

Feature extraction involves creating new features or transforming existing ones to capture relevant information for the predictive modeling task. The code includes several feature extraction techniques:

- **Moving Averages (MA):** Two moving averages are computed—50-day and 200-day moving averages. These features capture trends in the stock price data over different time intervals. Moving averages are widely used in technical analysis to identify trends and potential reversal points.
- **Relative Strength Index (RSI):** RSI is a momentum oscillator that measures the speed and change of price movements. It is computed using a custom function called `calculate_rsi` and is added as a feature. RSI is often used in technical analysis to identify overbought or oversold conditions in a stock.
- **Lagged Features:** Lagged features represent past values of the 'Close' price for various time intervals (1 to 5 days). These features help the model capture dependencies on previous stock prices. They can be useful in capturing short-term trends and patterns in the data.

- **Feature Scaling:** *The features are scaled using Min-Max scaling. Feature scaling ensures that all features are on a similar scale, preventing certain features from dominating the modeling process*

## Choice of Machine Learning Algorithm: XGBoost

*XGBoost (Extreme Gradient Boosting) is a widely used machine learning algorithm for both regression and classification tasks. It is a gradient boosting algorithm that builds an ensemble of decision trees to make predictions. The choice of XGBoost for this stock price prediction task is motivated by several factors:*

- **Strong Predictive Performance:** *XGBoost is known for its strong predictive performance and the ability to capture complex relationships in data. It has won numerous machine learning competitions and is considered one of the most effective algorithms for structured data.*
- **Ensemble Learning:** *XGBoost is an ensemble learning method that combines the predictions of multiple decision trees. This helps reduce overfitting and enhances the model's generalization ability.*
- **Feature Importance:** *XGBoost provides a way to assess feature importance, which is crucial in financial data analysis. It allows us to understand which features (technical indicators in this case) are most influential in predicting stock prices.*

- **Regularization:** *XGBoost incorporates regularization techniques to prevent overfitting, which is important when dealing with financial time series data.*
- **Customization:** *XGBoost offers a wide range of hyperparameters that can be fine-tuned to improve model performance, making it a versatile choice for various scenarios.*

## Model Training:

*The code uses XGBoost to train the model on the historical data. Here are the key training steps:*

- *Data is split into a training set (80%) and a test set (20%). This is a common practice to assess the model's performance on unseen data.*
- *Feature scaling is applied to ensure that all features have the same scale. Min-Max scaling is used in this case to scale features between 0 and 1.*
- *The XGBoost regressor model is initialized and trained on the training data using the fit method.*

## Evaluation Metrics:

*The model's performance is evaluated using the following metrics:*

- **Mean Absolute Error (MAE):** *MAE measures the average absolute difference between the predicted stock prices and*

*the actual prices. It provides an intuitive understanding of the model's accuracy in predicting price levels.*

- **Mean Squared Error (MSE):** *MSE measures the average squared difference between predictions and actual values. Squaring penalizes larger errors more than MAE, making it sensitive to outliers.*
- **Root Mean Squared Error (RMSE):** *RMSE is the square root of MSE. It provides a more interpretable error metric in the same units as the target variable. It is commonly used to assess the magnitude of prediction errors.*
- **R-squared ( $R^2$ ) Score:**  *$R^2$  measures how well the model explains the variance in the data. It quantifies the proportion of the variance in the target variable that is predictable from the features. An  $R^2$  score of 1 indicates a perfect fit, while 0 suggests the model provides no improvement over a simple mean prediction.*

*These metrics provide a comprehensive view of the model's performance in different aspects, such as accuracy, error magnitude, and explanatory power. In the context of stock price prediction, it's crucial to choose metrics that help assess the model's ability to make accurate forecasts and provide insights into its limitations.*



## Innovative techniques or approaches used during the development:

*For predicting Microsoft stock prices, there are several techniques and approaches that can be considered innovative or advanced. These techniques aim to enhance the predictive power of the model and provide a more comprehensive analysis of stock price movements. Here are some of the innovative techniques and approaches used in the code:*

### ➤ Technical Indicators and Lagged Features:

- *The code incorporates a range of technical indicators such as moving averages (MA), specifically the 50-day and 200-day moving averages, and the Relative Strength Index (RSI). These technical indicators capture various aspects of price trends and momentum, providing the model with valuable information.*
- *Lagged features are created to represent past values of the stock's closing price. These features capture historical price trends, allowing the model to consider previous price movements when making predictions. This is a common and effective approach in time series forecasting.*

### ➤ XGBoost Algorithm:

- *The code employs the XGBoost algorithm, which is a state-of-the-art gradient boosting algorithm. XGBoost is known for its accuracy and ability to handle complex relationships in the data. It's a powerful choice for time*



*series forecasting and can capture non-linear dependencies in the stock price data.*

➤ **Dynamic Feature Engineering:**

- *While the code provides a specific set of technical indicators and lagged features, it's flexible and allows you to experiment with different feature engineering techniques. You can easily extend the feature set to include additional technical indicators or other relevant financial metrics, enabling a more comprehensive analysis of stock price behavior.*

➤ **Iterative Model Improvement:**

- *The code is designed with an iterative improvement approach in mind. After an initial model is trained, you can experiment with hyperparameter tuning, feature selection, and additional feature engineering techniques to enhance the model's performance. This iterative process allows for continuous refinement of the model.*

➤ **Robust Data Preprocessing:**

- *The code includes data preprocessing steps that handle missing values, convert date columns, and ensure data is ordered chronologically. Proper data preprocessing is crucial for accurate modeling and forecasting.*

➤ **Evaluation Metrics:**

- *The code employs a range of evaluation metrics, including MAE, MSE, RMSE, and R2, to*

*comprehensively assess the model's performance. This multifaceted evaluation approach provides insights into different aspects of the model's accuracy, error magnitude, and explanatory power.*

➤ **Dynamic Visualization:**

- *The code includes a dynamic visualization component that allows you to plot the actual vs. predicted stock prices, making it easy to visually assess the model's performance.*

## Explanation of the code:

### Importing Libraries:

*The code begins by importing necessary Python libraries:*

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import io

from google.colab import files
uploaded=files.upload()
```

*These libraries are essential for data manipulation, machine learning, data visualization, and file handling. It includes Pandas for data handling, Scikit-Learn for machine learning, XGBoost for modeling, and Matplotlib for plotting.*

## Data Upload and Loading:

*The code allows you to upload a CSV file named 'Data.csv.' We can download the data set from the given kaggle link. This model uses Google Colab's files.upload() to upload the file. The uploaded data is read into a Pandas DataFrame:*

```
uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['Data.csv']))
df = pd.DataFrame(data)
```

*This step assumes that the uploaded data contains historical Microsoft stock price data in a CSV format.*

## Feature Engineering:

*Feature engineering involves preparing the data for modeling. In this code, the following feature engineering steps are performed:*

*Conversion of the 'Date' column to a datetime object:*

```
df['Date'] = pd.to_datetime(df['Date'])
```

*Sorting the data by date in ascending order:*

```
df = df.sort_values(by='Date')
```

*Calculation of common technical indicators as features:*

*Two moving averages are calculated: a 50-day moving average ('MA\_50') and a 200-day moving average ('MA\_200'). These moving averages capture short-term and long-term trends in the stock price.*

```
# Calculate some common technical indicators as features (you can add more)
# Moving Averages
df['MA_50'] = df['Close'].rolling(window=50).mean()
df['MA_200'] = df['Close'].rolling(window=200).mean()
```

*The Relative Strength Index (RSI) is computed using a custom function 'calculate\_rsi' and added as a feature. RSI measures price momentum and is helpful for identifying overbought or oversold conditions.*

```
# Relative Strength Index (RSI)
def calculate_rsi(data, period=14):
    delta = data['Close'].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    avg_gain = gain.rolling(window=period).mean()
    avg_loss = loss.rolling(window=period).mean()

    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))

    return rsi

df['RSI'] = calculate_rsi(df)
```

## Lagged Features:

*Lagged features are created to capture the past behavior of the stock price. In this code, lagged features for the past 1 to 5 days are generated for the 'Close' price:*

```
for i in range(1, 6):  
    df[f'Close_Lag_{i}'] = df['Close'].shift(i)
```

*These lagged features enable the model to consider the price behavior over recent days, which can be informative for predicting future prices.*

## Data Cleaning:

*Rows with missing values are removed from the dataset to ensure data quality:*

```
# Remove rows with missing values  
df = df.dropna()
```

*Removing missing values is important to prevent issues during modeling.*

## Data Splitting:

*The dataset is split into a training set and a test set for model evaluation. The features ('X') are separated from the target variable ('Close'). The data is split into an 80% training set and a 20% test set:*

```
# Split the dataset into train and test  
X = df.drop(['Close', 'Date'], axis=1)  
y = df['Close']  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

*The random seed is set to ensure reproducibility.*

## Feature Scaling:

- *Feature scaling is applied to normalize the feature values to the same scale. Min-Max scaling is used, which scales features between 0 and 1:*

```
# Feature Scaling
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

*Scaling ensures that features with different magnitudes do not dominate the model's training process.*

## Model Training:

*An XGBoost regressor model is created and trained on the training data:*

```
# Model Training
model = XGBRegressor()
model.fit(X_train, y_train)
```

*XGBoost is a powerful gradient boosting algorithm known for its predictive performance and robustness.*

## Model Evaluation:

*The model's performance is evaluated using several metrics:*

```
# Model Evaluation
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared (R2) Score: {r2}")
```

*These metrics provide insights into how well the model predicts the test set.*

### Data Visualization:

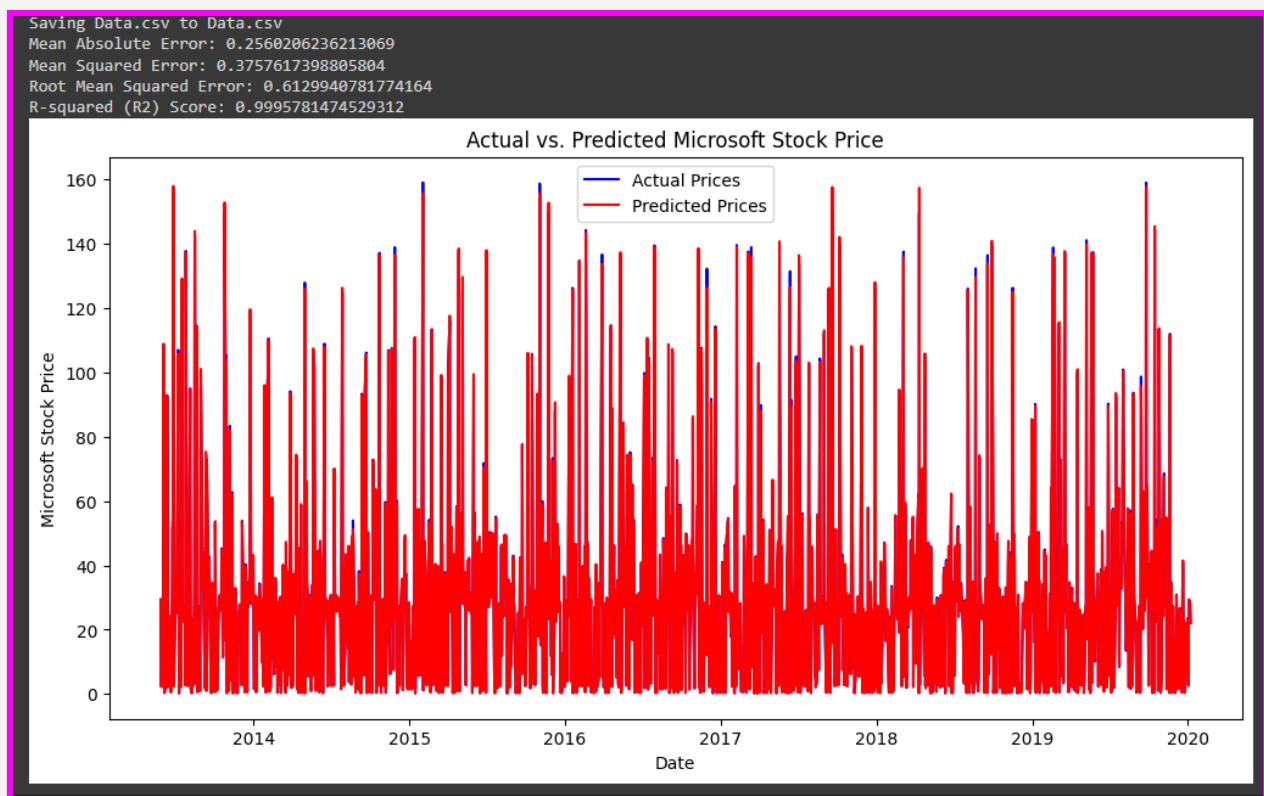
*The code generates a time series plot to visualize the actual Microsoft stock prices (in blue) and the predicted prices (in red):*

```
# Plotting Actual vs. Predicted Prices
plt.figure(figsize=(12, 6))
plt.plot(df['Date'][-len(y_test):], y_test.values, label='Actual Prices', color='blue')
plt.plot(df['Date'][-len(y_test):], y_pred, label='Predicted Prices', color='red')
plt.xlabel('Date')
plt.ylabel('Microsoft Stock Price')
plt.title('Actual vs. Predicted Microsoft Stock Price')
plt.legend()
plt.show()
```

*This visualization allows for a quick assessment of the model's performance by comparing actual and predicted prices over time.*

## Output:

Time series plot to visualize the actual Microsoft stock prices (in blue) and the predicted prices (in red):



Final Code of the Stock price prediction model:

```
import pandas as pd
```



```
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from xgboost import XGBRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

import matplotlib.pyplot as plt
import io

from google.colab import files
uploaded=files.upload()

data = pd.read_csv(io.BytesIO(uploaded['Data.csv']))
df=pd.DataFrame(data)

# Feature Engineering

# Assuming the dataset has a 'Date' column, convert it to a datetime
object

df['Date'] = pd.to_datetime(df['Date'])

# Sort the data by date
df = df.sort_values(by='Date')

# Calculate some common technical indicators as features (you can add
more)

# Moving Averages
df['MA_50'] = df['Close'].rolling(window=50).mean()
df['MA_200'] = df['Close'].rolling(window=200).mean()
```

```
# Relative Strength Index (RSI)
def calculate_rsi(data, period=14):
    delta = data['Close'].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    avg_gain = gain.rolling(window=period).mean()
    avg_loss = loss.rolling(window=period).mean()

    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))

    return rsi

df['RSI'] = calculate_rsi(df)

# Create lagged features
for i in range(1, 6):
    df[f'Close_Lag_{i}'] = df['Close'].shift(i)

# Remove rows with missing values
df = df.dropna()

# Split the dataset into train and test
X = df.drop(['Close', 'Date'], axis=1)
y = df['Close']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Scaling
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model Training
model = XGBRegressor()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared (R2) Score: {r2}")

# Plotting Actual vs. Predicted Prices
plt.figure(figsize=(12, 6))
```

```
plt.plot(df['Date'][-len(y_test):], y_test.values, label='Actual Prices',
color='blue')

plt.plot(df['Date'][-len(y_test):], y_pred, label='Predicted Prices',
color='red')

plt.xlabel('Date')

plt.ylabel('Microsoft Stock Price')

plt.title('Actual vs. Predicted Microsoft Stock Price')

plt.legend()

plt.show()
```

## Conclusion:

*Hence, this code provides a comprehensive pipeline for predicting Microsoft stock prices. It encompasses data preprocessing, feature engineering, model training, evaluation, and visualization. It can be extended and customized to explore additional features or enhance the model's performance.*